

Maximally Joining Probabilistic Data*

[Extended Abstract]

Benny Kimelfeld and Yehoshua Sagiv

The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University of Jerusalem
Edmond J. Safra Campus, Jerusalem 91904, Israel
{bennyk,sagiv}@cs.huji.ac.il

ABSTRACT

The common approach to manipulating probabilistic data is to evaluate relational queries and calculate the probability of each tuple in the result. It ignores the possibility that the probabilities of complete answers are too low and, hence, partial answers (with sufficiently high probabilities) become important. Thus, we propose a semantics that defines maximal answers (i.e., the degree of incompleteness is minimized) with respect to a given probability threshold.

We investigate the complexity of joining relations under the proposed semantics. In contrast to the deterministic case, this approach gives rise to two different *enumeration* problems. The first is finding all maximal *sets of tuples* that are join consistent, connected and have a probability above the threshold. The second is computing all maximal *tuples* that are answers of partial joins and have a probability above the threshold. Both problems are tractable under data complexity. Under query-and-data complexity, it becomes inefficient to compute all partial answers and then choose the maximal ones with respect to the given threshold. We give efficient algorithms for several important cases and show that, in general, the first problem is NP-hard whereas the second is #P-hard.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems—*Query processing, Relational databases*

General Terms

Algorithms

Keywords

Probabilistic databases, query-and-data complexity, maximal answers, natural join

*This research was supported by The Israel Science Foundation (Grant 893/05).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS'07, June 11–13, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-685-1/07/0006 ...\$5.00.

1. INTRODUCTION

The common approach to evaluating queries over probabilistic data is to apply the usual semantics of relational algebra and compute the probability of each tuple in the result [4, 10, 9, 11, 13, 18, 1]. This approach may produce answers with probabilities that are too low to be of interest. When data have probabilities, there is a tradeoff between complete answers (i.e., without null values) that have rather low probabilities and partial answers that carry less information but have higher probabilities. In ordinary databases, by comparison, there is no such tradeoff, because if a complete answer subsumes a partial one, then the latter is deemed meaningless. But when probabilities are taken into account, partial answers may have much higher probabilities and, hence, be more meaningful. We illustrate this phenomenon with a few examples.

Consider the three relations of Figure 1 (but ignore column E for now). Note that each tuple is identified as t_i , where $1 \leq i \leq 10$. Suppose that the query is the natural join of the three relations ($Person$ is the only common attribute, since column E is ignored). A *complete* answer is obtained by joining three tuples, one from each relation; for example, $t_1 \bowtie t_4 \bowtie t_7$ is a complete answer. A *partial* answer is a join of fewer than three tuples, e.g., $t_1 \bowtie t_7$. We are interested in a partial answer if it is *maximal*, namely, not subsumed (i.e., not part of) another answer that carries more information. For example, the partial answer $t_1 \bowtie t_7$ is not maximal, because it is subsumed by $t_1 \bowtie t_4 \bowtie t_7$. Since there is no tuple with the value “Johnson” in the **Professors** relation, $t_2 \bowtie t_8$ is a maximal but not a complete answer.

Now, suppose that we take probabilities into account. In Figure 1, the probability of each tuple is given in column E inside parentheses. The complete answer $t_1 \bowtie t_4 \bowtie t_7$ has the probability $0.9 \cdot 0.5 \cdot 0.9 = 0.405$ (i.e., the product of the probabilities of the three tuples) whereas the probability of the partial answer $t_1 \bowtie t_7$ is $0.9 \cdot 0.9 = 0.81$, which is much higher. If the underlying semantics of queries is the usual one (i.e., answers are always complete) and the user is interested only in answers with a probability of at least 0.5, she will not get any information about Smith.

The above example shows that partial answers must be taken into account in order to provide users with meaningful results. One may think that it can be done by computing maximal answers and finding the ones that have sufficiently high probabilities. However, this straightforward approach does not really solve the problem, as shown by the next example (assuming, once again, that we are interested only in answers that have a probability of at least 0.5). Both $t_1 \bowtie t_7$

and $t_2 \bowtie t_8$ have probabilities (0.81 and 0.64, respectively) above the threshold of 0.5, but only the latter is maximal whereas the former is subsumed by $t_1 \bowtie t_4 \bowtie t_7$, which has the probability 0.405. Hence, the user will get the maximal answer $t_2 \bowtie t_8$ about Johnson, but no answer at all about Smith. Yet, $t_1 \bowtie t_7$ supplies exactly the same type of information about Smith, with even a higher probability, as $t_2 \bowtie t_8$ does about Johnson.

It thus follows that probabilities should play a role in defining maximal answers, because merely superimposing probabilities on an existing semantics for maximal answers (e.g., [16, 6, 14]) creates unexpected anomalies. In other words, in a probabilistic data model, it only makes sense to evaluate queries with respect to a *threshold* (specified by the user) that determines the minimal amount of certainty that answers should have. The threshold is actually a tool for defining maximal answers and not merely a filter for selecting answers that pass a certain level of certainty among those that are plainly maximal (according to a notion that does not incorporate probabilities).

Thus, we define maximal answers as follows. We assume that the user specifies a threshold p ($0 \leq p \leq 1$). When joining several relations, we consider all partial answers that have a probability of at least p and select the maximal ones. For example, if $p = 0.5$, then $t_1 \bowtie t_7$ is a maximal answer. But if we choose $p = 0.4$, then $t_1 \bowtie t_4 \bowtie t_7$ is maximal whereas $t_1 \bowtie t_7$ is not.

We propose a model of probabilistic relational databases (p-rdb) in which every tuple is associated with an atomic event. An *instance* I of a p-rdb is a regular database that corresponds to a set of events V (and comprises the tuples produced by those events). The *probability* of I is the same as the probability that exactly the events of V occur. In our model, an event can produce multiple tuples in several relations and, moreover, some events can be *mutually exclusive*, i.e., only one them can occur. Thus, our model is similar to, yet more general than that of [10], but they analyze evaluation of general SQL queries under data complexity.

We investigate two types of partial answers. One is based on tuples that are produced by partial joins whereas the second involves sets of tuples from the database that yield results of the first type. In the database of Figure 1, there is a one-to-one correspondence between the partial answers of these two types. But sometimes different sets of tuples from the database can produce the same result. For example, consider the database of Figure 2. When viewing results of partial joins, $t_1 \bowtie t_2$ and $t_1 \bowtie t_3$ are the same. But the sets $\{t_1, t_2\}$ and $\{t_1, t_3\}$ are different *witnesses* of the result u_7 (where $u_7 = t_1 \bowtie t_2 = t_1 \bowtie t_3$). When defining maximal answers in regular databases, these two types of partial answers lead to the same concept of *full disjunction* [14], which is tractable [17, 7, 5]. But when probabilities are taken into account, there could be a difference, because u_7 is likely to have a higher probability than that of either $\{t_1, t_2\}$ or $\{t_1, t_3\}$. The bottom line is that computing maximal sets of tuples is more tractable. Arguably, it also seems more suitable in practice, since it is important to know how results of partial joins are derived rather than merely getting them.

We show that under data complexity, both types of maximal answers can be computed efficiently. However, query-and-data complexity is the real measure of efficiency, since computing all partial answers and then selecting those that have a probability above the threshold is hardly a practical

approach. In addition, the running time should be computed as a function of the combined size of the input and the output [15], since there could be exponentially-many maximal answers (in the size of the input).

Generally, computing either type of maximal answers is intractable, under query-and-data complexity. Nevertheless, in Sections 4, we show that computing maximal sets of tuples is tractable in several, practically significant cases. The most important one is γ -acyclic databases [2, 3, 12, 19, 25] in which the same event does not occur in different relations (each of the two conditions alone does not make the problem tractable). Other cases are database schemes, such that every two relations have some shared attributes (e.g., Figures 1 and 2), and databases with special types of probabilities.

In Section 5, we show that in all of the above special cases, the problem of computing maximal tuples (that are produced by partial joins) remains intractable. One approach to overcome this difficulty is to identify cases where both types of maximal answers are equivalent. Clearly, one such case is when every relation has a unique attribute. Another case is γ -acyclic database schemes without containments among the relation schemes. It thus follows that this case is tractable if the same event does not occur in different relations. The same is true for tree schemes, even if there are containments among the relation schemes.

Our tractability results are derived from an algorithm (given in Sections 4) that reduces the problem of computing maximal sets of tuples to a related decision problem. Rather surprisingly, we are able to show that this reduction is optimal in the following sense. If, for a given class of schemes, the decision problem is intractable, then no algorithm can efficiently enumerate maximal sets of tuples (unless $P=NP$). Thus, no algorithm captures more classes of database schemes than ours.

2. PRELIMINARIES

2.1 Relations, Databases and Schemes

We define *relations*, *tuples* and *attributes* in the usual way. $\text{Attr}(R)$ and $\text{Attr}(t)$ denote the sets of attributes of the relation R and the tuple t , respectively. Given an attribute $A \in \text{Attr}(t)$, we use $t[A]$ to denote the value of t for A . A *relational database* (*rdb*) is a collection of relations. By a slight abuse of notation, we use $t \in D$ to denote the fact that the tuple t belongs to one of the relations of the rdb D . Since relations are just sets of tuples, $R' \subseteq R$ simply means that the relation R' is obtained by deleting zero or more tuples from the relation R . Similarly, $D' \subseteq D$ denotes that the rdb D' is the result of deleting zero or more tuples of the rdb D .

A *database scheme* (or just *scheme*) is a sequence $S = (\mathcal{A}_1, \dots, \mathcal{A}_k)$, where each \mathcal{A}_i is a set of attributes. We say that the rdb D is *over* S if it consists of k relations R_1, \dots, R_k , such that $\text{Attr}(R_i) = \mathcal{A}_i$ for all $1 \leq i \leq k$. The set \mathcal{A}_i is called the *relation scheme* (or just *scheme*) of R_i . Note that two distinct relations may have the same scheme and, hence, a database scheme is a sequence rather than a set. If, indeed, there are multiple relations with the same set of attributes, then we assume that each one has a unique name, in addition to its scheme.

The *join graph* of a database scheme S , denoted by $\mathcal{G}(S)$, is an undirected graph that has a distinct node n_i for each \mathcal{A}_i and has an edge between every two nodes n_i and n_j , such

Researchers			
	Person	Company	E
t_1	Smith	Yahoo	$e_1(0.9)$
t_2	Johnson	IBM	$e_2(0.8)$
t_3	Adams	Google	$e_{3a}(0.5)$

Professors			
	Person	Institute	E
t_4	Smith	MIT	$e_4(0.5)$
t_5	Adams	Princeton	$e_{5a}(0.1)$
t_6	Adams	Berkeley	$e_{5b}(0.05)$

Projects				
	Person	Project	Role	E
t_7	Smith	TagLines	Res.	$e_6(0.9)$
t_8	Johnson	Clio	Res.	$e_7(0.8)$
t_9	Adams	G. Base	Res.	$e_{3a}(0.5)$
t_{10}	Adams	TagLines	Man.	$e_{3b}(0.4)$

Distributions (X)	
x_1	$e_1(0.9)$
x_2	$e_2(0.8)$
x_3	$e_{3a}(0.5), e_{3b}(0.4)$
x_4	$e_4(0.5)$
x_5	$e_{5a}(0.1), e_{5b}(0.05)$
x_6	$e_6(0.9)$
x_7	$e_7(0.8)$

Figure 1: A p-rdb \mathcal{P} with the set of relations D^E (on the left) and the set of distributions X (on the right)

that $1 \leq i < j \leq k$ and $\mathcal{A}_i \cap \mathcal{A}_j \neq \emptyset$. We assume that $\mathcal{G}(S)$ is connected, or else all the problems that we address can be solved for each connected component separately.

We consider three specific classes of database schemes. $\mathcal{S}_{\text{clique}}$ is the class of all database schemes S , such that $\mathcal{G}(S)$ is a clique. For example, if there is an attribute that appears in all the relations of the database, then the scheme is in $\mathcal{S}_{\text{clique}}$. The class $\mathcal{S}_{\text{tree}}$ consists of all database schemes S , such that $\mathcal{G}(S)$ is a tree. Finally, \mathcal{S}_γ is the class of all γ -acyclic database schemes [12]. Note that $\mathcal{S}_{\text{tree}} \subsetneq \mathcal{S}_\gamma$.

2.2 Enumeration Algorithms

An *enumeration algorithm Alg* generates, for a given input y , a sequence a_1, \dots, a_m of *answers*. Each a_i is produced by the operation $\text{print}(\cdot)$. We say that $\text{Alg}(y)$ *enumerates* the set M if, during the execution of $\text{Alg}(y)$, every element of M is produced exactly once. Next, we discuss measures of efficiency for enumeration algorithms.

Polynomial time is not a suitable yardstick of efficiency when analyzing an enumeration algorithm, since the size of the output could be much larger (e.g., exponentially larger) than the size of the input. In [15], several definitions of efficiency for enumeration algorithms are discussed. The weakest definition is *polynomial total time*, where the running time is polynomial in the combined size of the input and the output. Two stronger definitions consider the time that is needed for generating the i th element, after the first $i - 1$ elements have already been created. *Incremental polynomial time* means that the i th element is generated in time that is polynomial in the combined size of the input and the first $i - 1$ elements. The strongest definition is *polynomial delay*, which means that the i th element is generated in time that is polynomial only in the input.

3. PROBABILISTIC RDBS

3.1 Modeling Probabilistic Data

Probabilistic data is produced by *events*. We use the attribute E to specify the event e_i that creates each tuple (see Figure 1). The tuples produced by each event are predetermined, but the events themselves are randomly generated according to some given distributions. By a slight abuse of terminology, we view a distribution as a finite set of events. Formally, a *distribution* x assigns a probability $\Pr(e) \in (0, 1]$ to each event $e \in x$. We assume that $\sum_{e \in x} \Pr(e) \leq 1$ and denote the sum $\sum_{e \in x} \Pr(e)$ by $\Pr(x)$. By definition, each distribution x is *mutually exclusive* in the sense that it gen-

erates at most one event. That is, the joint probability of two or more events of x is 0. Therefore, the probability that no event of x occurs is $1 - \Pr(x)$. Different distributions are independent and pairwise disjoint (i.e., do not contain the same event).

A *probabilistic rdb (p-rdb)* is a pair $\mathcal{P} = (D^E, X)$, such that D^E is a set of relations and X is a set of distributions. The superscript E emphasizes that every relation R^E of D^E uses the special attribute E in order to specify the event that produces each tuple. We say that an attribute is *ordinary* if it is other than E . When talking about the database scheme of a p-rdb, we only consider the ordinary attributes (and ignore the special attribute E).

As an example, Figure 1 shows a p-rdb $\mathcal{P} = (D^E, X)$ with three relations and a set of distributions X (on the right side). Next to each event, the probability is shown inside parentheses. For example, e_1 is the only event of the distribution x_1 and it has the probability 0.9. The distributions x_2, x_4 and x_7 are also singletons, while each of x_3 and x_5 contains two events. So, for example, e_{3a} and e_{3b} are mutually exclusive, yet e_{5a} and e_{3a} are independent.

The ability of an event to generate multiple tuples, in one or more relations, is essential for modeling probabilistic data properly. The reason is that for some given pieces of information, we may know that either they exist together or none of them is valid. For example, the information about Adams in tuples t_3 and t_9 was presumably obtained from the same source and, hence, these tuples are generated by the same event e_{3a} that has the probability 0.5.

An attempt to model conditional probabilities is rather difficult and, so, the best that we can do is to assume that different sources of information are independent. For example, the information about Smith in each of the tuples t_1, t_4 and t_7 was obtained from a different source and, hence, these tuples are generated by three distinct events.

However, sometimes we may know that two pieces of data cannot both be valid, even if they are collected from different sources. For example, the same person cannot be an employee of two competing companies at the same time. In such situations, mutual exclusion becomes an important modeling tool. For example, Adams cannot work for Yahoo and Google at the same time. So, the event e_{3a} generates the tuples t_3 and t_9 that pertain to his work in Google while a second event from the same distribution, e_{3b} , creates the tuple t_{10} that is about his work in a project of Yahoo.

We say that a p-rdb is *localized* if each event appears in only one relation (where it can have multiple occurrences). For example, the p-rdb \mathcal{P} of Figure 1 is not localized, since

e_{3a} appears in two different relations. \mathcal{P} would become localized if the tuple t_9 was removed. A stronger notion is *relation independence*, which means that all the events of the same distribution can generate tuples in only one relation (hence, if two events appear in different relations, then they cannot belong to the same distribution and, thus, are independent). If all the distributions are singletons, then localization and relation independence are the same.

The probabilistic data model of [10] does not allow mutual exclusion (i.e., all distributions are singletons) and assumes relation independence. Hence, it is more restricted than our framework.

3.2 Instances

Consider a p-rdb $\mathcal{P} = (D^E, X)$. We use $\pi_{\perp E}$ to denote the operation of projecting out (i.e., erasing) column E . We apply $\pi_{\perp E}$ to a relation or a tuple of \mathcal{P} , as well as to sets of relations or sets of tuples.

A set of events S is *feasible* if it has at most one event from each distribution of X . We say that \hat{D}^E is a *p-instance* of \mathcal{P} if there is a feasible set of events S , such that \hat{D}^E comprises all the tuples produced by the events of S (hence, $\hat{D}^E \subseteq D^E$). If \hat{D}^E is a p-instance, then $I = \pi_{\perp E}(\hat{D}^E)$ is an *ordinary instance* (or just *instance* for short). $\mathcal{I}(\mathcal{P})$ denotes the set of all instances (i.e., *possible worlds*) of \mathcal{P} .

We assume that an instance I corresponds to exactly one p-instance \hat{D}^E and, hence, to a unique feasible set of events. Concretely, it means that distinct events cannot generate identical tuples in the same relation. Formally, for all relations R^E of D^E , there are no duplicates in the relation $\pi_{\perp E}(R^E)$. In other words, R^E cannot have distinct tuples that are equal on all their ordinary attributes.

Two distinct relations of D^E may have the same set of attributes. If so, tuples are augmented with their relation names, thereby ensuring that tuples from different relations are always distinct entities. Thus, for each tuple of D^E , the value in column E is uniquely determined by the values of the other attributes (and the relation name, if necessary).

Let S be a feasible set of events. The probability that exactly the events of S occur is

$$\prod_{e \in S} \Pr(e) \prod_{(x \in X) \wedge (x \cap S = \emptyset)} (1 - \Pr(x))$$

i.e., the product of the probabilities of the events of S and the probabilities that distributions without representatives in S generate no events. We also call it the probability of S . The probability of an instance $I \in \mathcal{I}(\mathcal{P})$, denoted by $\Pr(I)$, is the same as the probability of the unique feasible set of events that corresponds to I .

4. MAXIMAL JCC TUPLE SETS

Consider a p-rdb $\mathcal{P} = (D^E, X)$. Let t be a tuple of an instance $I \in \mathcal{I}(\mathcal{P})$. We use t^E to denote the unique tuple of D^E that corresponds to t . Similarly, if T is a set of tuples of I , then T^E is the set of the corresponding tuples of D^E . Tuples and sets of tuples of D^E are usually (but not always) written as t^E and T^E , respectively; however, even if E is omitted, t and T are the same as t^E and T^E , respectively.

We use Δ to denote a set of relations that is either D^E or an instance of \mathcal{P} . Two tuples t_1 and t_2 of Δ are *join consistent* if they are equal on every ordinary attribute (i.e., other than E) that appears in both of them (hence, they cannot

be from the same relation). A *tuple set* T of Δ is any set of tuples from Δ . The tuple set T is *join consistent* if every two tuples of T are join consistent and, moreover, the set of events that appear in T , denoted by $\pi_E(T)$, is feasible. (Note that the latter part is nontrivial only if Δ is D^E ; otherwise, it always holds.) The tuple set T is *connected* if its *join graph* $\mathcal{G}(T)$ is connected. $\mathcal{G}(T)$ is an undirected graph with nodes that correspond to the tuples of T and edges that connect pairs of tuples that have at least one ordinary attribute in common.¹ Finally, T is *JCC* if T is both join consistent and connected.

Example 1. In the p-rdb \mathcal{P} of Figure 1, both $\{t_1, t_4\}$ and $\{t_1, t_4, t_7\}$ are JCC tuple sets. But $\{t_1, t_4, t_8\}$ is not a JCC tuple set, since t_1 and t_8 (and also t_4 and t_8) do not agree on the attribute *Person*. The set $\{t_3, t_5, t_9\}$ is JCC, but $\{t_3, t_5, t_{10}\}$ is not because t_3 and t_{10} have mutually exclusive events.

$\mathcal{C}(\Delta)$ denotes the set of all JCC tuple sets of Δ . Let $\mathcal{T} \subseteq \mathcal{C}(\Delta)$. T is a *maximal* tuple set of \mathcal{T} if $T \in \mathcal{T}$ and T is not properly contained in any other tuple set of \mathcal{T} .

If $\Delta = D^E$, then we usually write $\mathcal{C}(\mathcal{P})$ instead of $\mathcal{C}(D^E)$. Similarly, if $T^E \in \mathcal{C}(\mathcal{P})$, then we say that T^E is a JCC tuple set of \mathcal{P} . If T is a maximal tuple set of $\mathcal{C}(\mathcal{P})$, then T is just *maximal*.

Example 2. Consider the p-rdb \mathcal{P}_1 of Figure 2. The second table from the right shows $\mathcal{C}(\mathcal{P}_1)$. Each tuple set is identified as T_i ($1 \leq i \leq 12$) and also written explicitly in the middle column. Note that the first row of the table defines T_i for $i = 1, \dots, 6$. (The column $\Pr(T)$ gives the probability of each tuple set, as explained in the next section.) Let $\mathcal{T} = \{T_1, \dots, T_9\}$. The maximal tuple sets of \mathcal{T} are T_4, T_5, T_6 and T_9 . The maximal tuple sets of $\mathcal{C}(\mathcal{P}_1)$ are T_6, T_9, T_{11} and T_{12} .

4.1 Probable JCC Tuple Sets

Observe the following. If T is a JCC tuple set of an instance $I \in \mathcal{I}(\mathcal{P})$, then T^E is a JCC tuple set of D^E . Conversely, if T^E is a JCC tuple set of D^E , then $\pi_{\perp E}(T^E)$ is a JCC tuple set of one or more instances of $\mathcal{I}(\mathcal{P})$.

Consider a JCC tuple set T^E of \mathcal{P} , i.e., $T^E \in \mathcal{C}(\mathcal{P})$. We define the probability of T^E , denoted by $\Pr(T^E)$, as the probability that a random instance of $\mathcal{I}(\mathcal{P})$ has $\pi_{\perp E}(T^E)$ as one of its JCC tuple sets. The following proposition is straightforward.

PROPOSITION 4.1. *Consider a JCC tuple set T^E of \mathcal{P} . Let $S = \pi_E(T^E)$. Then, $\Pr(T^E) = \prod_{e \in S} \Pr(e)$.*

As an example, Figure 2 shows the probability of each tuple set of the p-rdb \mathcal{P}_1 in the column labeled with $\Pr(T)$.

Given a *threshold* $p \in [0, 1]$, we use $\mathcal{C}_{\geq p}(\mathcal{P})$ to denote the set $\{T^E \mid T^E \in \mathcal{C}(\mathcal{P}) \text{ and } \Pr(T^E) \geq p\}$. $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$ denotes the set of all the maximal elements of $\mathcal{C}_{\geq p}(\mathcal{P})$.

Example 3. Consider the p-rdb \mathcal{P}_1 of Figure 2. If the threshold is 0.5, then $\mathcal{C}_{\geq 0.5}(\mathcal{P}_1) = \{T_1, \dots, T_5, T_{12}\}$ and $\mathcal{C}_{\geq 0.5}^{\max}(\mathcal{P}_1) = \{T_1, T_2, T_3, T_{12}\}$. For the threshold of 0.4, we get that $\mathcal{C}_{\geq 0.4}(\mathcal{P}_1) = \{T_1, \dots, T_5, T_7, T_8, T_{10}, T_{11}, T_{12}\}$ and $\mathcal{C}_{\geq 0.4}^{\max}(\mathcal{P}_1) = \{T_7, T_8, T_{10}, T_{11}, T_{12}\}$.

DEFINITION 4.2. *ProbableJCC is the problem of enumerating $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$, given a p-rdb \mathcal{P} and a threshold $p \geq 0$.*

¹Some of our results hold also for other notions of join consistency and connectivity, e.g., equijoin or theta-join.

D^E				X		$\mathcal{C}(\mathcal{P}_1)$		$\mathcal{J}(\mathcal{P}_1)$						
A	B	C	E	x_1	x_2	T	$\Pr(T)$	u	A	B	C	D	$\Pr(u)$	
t_1	1	1	1	$e_1(0.7)$	$e_2(0.7)$	T_i	$\{t_i\}$	u_1	1	1	1	\perp	0.7	
t_4	1	2	2	$e_{4a}(0.6)$	$e_3(0.7)$	T_7	$\{t_1, t_2\}$	0.49	u_2	1	\perp	\perp	1	0.7
					$e_4(0.6), e_{4b}(0.3)$	T_8	$\{t_1, t_3\}$	0.49	u_3	\perp	1	\perp	1	0.7
						T_9	$\{t_1, t_2, t_3\}$	0.343	u_4	1	2	2	\perp	0.6
						T_{10}	$\{t_2, t_3\}$	0.49	u_5	\perp	2	\perp	2	0.6
						T_{11}	$\{t_2, t_4\}$	0.42	u_6	\perp	2	\perp	3	0.3
						T_{12}	$\{t_4, t_5\}$	0.6	u_7	1	1	1	1	0.637
									u_8	1	1	\perp	1	0.49
									u_9	1	2	2	1	0.42
									u_{10}	1	2	2	2	0.6

Figure 2: A p-rdb \mathcal{P}_1 and the sets $\mathcal{C}(\mathcal{P}_1)$ and $\mathcal{J}(\mathcal{P}_1)$

Algorithm PROBABLEJCC(\mathcal{P}, p)	
1:	$\mathcal{C}, \mathcal{Q} \leftarrow \emptyset$
2:	$T_0^E \leftarrow \text{MAXIMIZE}(\mathcal{P}, p, \emptyset)$
3:	print (T_0^E)
4:	$\mathcal{Q}.\text{INSERT}(T_0^E)$
5:	while $\mathcal{Q} \neq \emptyset$ do
6:	$T^E \leftarrow \mathcal{Q}.\text{REMOVE}()$
7:	$\mathcal{C}.\text{INSERT}(T^E)$
8:	for all tuples $t^E \in \mathcal{P}$ do
9:	$T_m^E \leftarrow \text{MAXJCCSUBSET}(\mathcal{P}, T^E, t^E)$
10:	for all $\hat{T}^E \in \text{MxEVCVRS}(\mathcal{P}, p, T_m^E, \{t^E[E]\})$ do
11:	$\hat{T}^E \leftarrow \text{MAXIMIZE}(\mathcal{P}, p, \hat{T}^E)$
12:	if $\hat{T}^E \notin \mathcal{Q} \cup \mathcal{C}$ then
13:	print (\hat{T}^E)
14:	$\mathcal{Q}.\text{INSERT}(\hat{T}^E)$

Figure 3: Computing $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$

4.2 The Complexity of ProbableJCC

In this section, we consider the query-and-data complexity of *ProbableJCC*. (Proposition 4.1 implies that *ProbableJCC* is tractable under data complexity.) Our first result shows how to reduce the enumeration problem *ProbableJCC* to the decision problem *EventCover* that is defined as follows.

Consider a p-rdb \mathcal{P} and a threshold $0 \leq p \leq 1$. Let T^E be a JCC tuple set of \mathcal{P} and ev be a feasible set of events. We say that T^E covers ev if $ev \subseteq \pi_E(T^E)$. We denote by $\mathcal{C}_{\geq p}^{\geq ev}(T^E)$ the set of all JCC subsets T_c^E of T^E , such that T_c^E covers ev and $\Pr(T_c^E) \geq p$.

Example 4. Consider the JCC tuple set T_9 of \mathcal{P}_1 (Figure 2). Suppose that $ev = \{e_2\}$. For the threshold of 0.5, only the JCC subset $\{t_2\}$ of T_9 is in $\mathcal{C}_{\geq 0.5}^{\geq ev}(T_9)$. If the threshold is 0.4, then $\mathcal{C}_{\geq 0.4}^{\geq ev}(T_9) = \{\{t_1, t_2\}, \{t_2, t_3\}\}$. If $ev = \{e_1, e_2\}$, then $\mathcal{C}_{\geq 0.5}^{\geq ev}(T_9)$ is empty and $\mathcal{C}_{\geq 0.4}^{\geq ev}(T_9) = \{\{t_1, t_2\}\}$.

DEFINITION 4.3. *EventCover* is the problem of deciding, given a JCC tuple set T^E , a set of events ev and a threshold $p \geq 0$, whether $\mathcal{C}_{\geq p}^{\geq ev}(T^E) \neq \emptyset$.

4.2.1 The Algorithm

If all events have probability 1, then $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$ is the same as the full-disjunction operator. In other words, $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$

consists of all the maximal JCC tuple sets, i.e., maximal elements of $\mathcal{C}(\mathcal{P})$. So, our approach is to use the algorithm of [5] that is based on the following idea. If T is a maximal JCC tuple set, then another maximal JCC tuple set can be obtained from T as follows. Choose a tuple t that is not in T and do the following. First, remove from $T \cup \{t\}$ all tuples that are not join consistent with t . Second, consider the join graph of the remaining tuples, and let \bar{T} be the set of all the tuples that appear in the same connected component as t . We can now apply a greedy algorithm that iterates over all tuples \bar{t} of the database and adds \bar{t} to \bar{T} if both join consistency and connectedness are preserved. The result is a maximal JCC tuple set \hat{T} . The above idea can be applied again to T (with a tuple other than t) and also to \hat{T} , in order to get more maximal JCC tuple sets.

When the probabilities are not necessarily 1, we cannot add \bar{t} to \bar{T} if the probability of $\bar{T} \cup \{\bar{t}\}$ is below the threshold. Moreover, the probability of the initial \bar{T} may already be below the threshold. Nevertheless, if we first find all maximal JCC subsets of \bar{T} that have a probability of at least p , then we can continue with the original approach by greedily extending each one of these subsets (but as we show later, finding these maximal subsets is not a straightforward task). The exact details are given below.

The algorithm *PROBABLEJCC*(\mathcal{P}, p) of Figure 3 computes $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$. (We assume that this algorithm is applied only if the p-rdb has a tuple with a probability of at least p , or else the result is empty.) It uses two repositories, \mathcal{C} and \mathcal{Q} , that store elements of $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$. (Operations on both \mathcal{C} and \mathcal{Q} take logarithmic time in the number of elements.) \mathcal{C} holds the tuple sets that have already been printed and processed (i.e., used in order to generate other elements of $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$) whereas \mathcal{Q} stores tuple sets that have been printed but not yet processed. Both \mathcal{C} and \mathcal{Q} are initially empty. Line 2 executes the greedy algorithm *MAXIMIZE*($\mathcal{P}, p, \emptyset$) that extends the empty set into an arbitrary element T_0^E of $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$, by repeatedly adding tuples as long as the result is in $\mathcal{C}_{\geq p}(\mathcal{P})$. In Lines 3–4, T_0^E is printed and then inserted into \mathcal{Q} . Lines 6–14 are repeatedly executed until \mathcal{Q} is empty. We first remove an element T^E from \mathcal{Q} and place it in \mathcal{C} (Lines 6–7). Next, the loop of Lines 9–14 is executed for all tuples t^E of \mathcal{P} . In Line 9, we apply the algorithm *MAXJCCSUBSET*(\mathcal{P}, T^E, t^E) that computes the maximal JCC subset T_m^E of $T^E \cup \{t^E\}$ that contains t^E , as explained in the first paragraph of this section (the pseudo code is omitted). Note, however, that the probability of T_m^E may be smaller than p . So, this is the

point where our algorithm becomes substantially different from that of [5].

Lines 11–14 are executed for all maximal subsets \bar{T}^E of T_m^E , such that $t^E \in \bar{T}^E$ and $\Pr(\bar{T}^E) \geq p$. These subsets are exactly the same as the maximal tuple sets of $\mathcal{C}_{\geq p}^{\supseteq ev}(T_m^E)$, where ev is the singleton containing the event of t^E . Thus, the algorithm `MxEVCVRS`, which is described below, is used for enumerating $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T_m^E))$, i.e., the set of maximal elements of $\mathcal{C}_{\geq p}^{\supseteq ev}(T_m^E)$.

In Line 11, each \bar{T}^E is extended to an arbitrary tuple set $\hat{T}^E \in \mathcal{C}_{\geq p}^{\max}(\mathcal{P})$ by calling `MAXIMIZE`($\mathcal{P}, p, \bar{T}^E$). Finally, if \hat{T}^E is in neither \mathcal{Q} nor \mathcal{C} (i.e., the test of Line 12 is **true**), then \hat{T}^E is printed and inserted into \mathcal{Q} . The following lemma shows the correctness and efficiency of the algorithm `PROBABLEJCC`.

LEMMA 4.4. *If the subroutine `MxEVCVRS`(\mathcal{P}, p, T^E, ev) enumerates $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T^E))$ in incremental polynomial time, then the algorithm `PROBABLEJCC`(\mathcal{P}, p) computes $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$ in incremental polynomial time.*

We now describe the algorithm `MxEVCVRS`(\mathcal{P}, p, T^E, ev) of Figure 4 for enumerating $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T^E))$, where T^E is a JCC tuple set of \mathcal{P} , ev is a set of events and $0 \leq p \leq 1$. This algorithm employs an oracle (which is discussed later) for deciding whether $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E)$ is empty. The oracle is applied in Line 1 and if the test is **true**, then the algorithm returns the empty result. Note that, in particular, if T^E does not cover ev , then the test is **true**. In fact, just testing whether T^E does not cover ev is sufficient for correctness, but not for achieving an enumeration with polynomial delay.

Line 2 tests whether $\Pr(T^E) \geq p$ and if so, then T^E itself is the only maximal tuple set of $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E)$. So, the algorithm terminates after printing T^E . Line 6 chooses an event e_m that has the minimal probability among the events that appear in T^E but not in ev . Now, consider the tuple set $T_{-e_m}^E$ that is obtained from T^E by removing all the tuples that have the event e_m . Clearly, $T_{-e_m}^E$ is join consistent but not necessarily connected. So, let T_1^E, \dots, T_k^E be the partition of $T_{-e_m}^E$ into connected components, according to the join graph $\mathcal{G}(T_{-e_m}^E)$. Note that a maximal element of $\mathcal{C}_{\geq p}^{\supseteq ev}(T_i^E)$, where $1 \leq i \leq k$, is also a maximal JCC tuple set of $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E)$ if and only if the test of Line 11 is **true**, i.e., $\Pr(T_i^E) \cdot \Pr(e_m) < p$. (If this test is **false**, then there is at least one tuple $t^E \in T^E$ with the event e_m , such that $T_i^E \cup \{t^E\}$ is in $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E)$.) Thus, Lines 12 applies the algorithm recursively to each T_i^E , such that the test of Line 11 is **true**, in order to generate the elements of $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T_i^E))$. It can be shown that the loop of Lines 10–12 enumerates all the elements of $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T^E))$ that do not include the event e_m . So, in Line 13, the remaining elements are enumerated by recursively executing the algorithm after adding the event e_m to ev . The correctness and efficiency of the algorithm are shown in the next lemma. The proof is by an induction on the number of events in T^E that are not in ev , i.e., the size of $\pi_E(T^E) \setminus ev$.

LEMMA 4.5. *`MxEVCVRS`(\mathcal{P}, p, T^E, ev) enumerates the elements of $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T^E))$. Furthermore, the delay is polynomial if Line 1 takes polynomial time.*

By Lemmas 4.4 and 4.5, the enumeration of $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$ can

Algorithm `MxEVCVRS`(\mathcal{P}, p, T^E, ev)

```

1: if  $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E) = \emptyset$  then
2:   return
3: if  $\Pr(T^E) \geq p$  then
4:   print( $T^E$ )
5:   return
6:    $e_m \leftarrow$  an event  $e \in \pi_E(T^E) \setminus ev$ , s.t.  $\Pr(e)$  is minimal
7:    $T_{-e_m}^E \leftarrow \{t \in T^E \mid t[E] \neq e_m\}$ 
8:    $G_1, \dots, G_k \leftarrow$  the connected components of  $\mathcal{G}(T_{-e_m}^E)$ 
9:    $T_1^E, \dots, T_k^E \leftarrow$  the sets of nodes of  $G_1, \dots, G_k$ , resp.
10:  for  $i = 1$  to  $k$  do
11:    if  $\Pr(T_i^E) \cdot \Pr(e_m) < p$  then
12:      MxEVCVRS( $\mathcal{P}, p, T_i^E, ev$ )
13:  MxEVCVRS( $\mathcal{P}, p, T^E, ev \cup \{e_m\}$ )

```

Figure 4: Enumerating $\max(\mathcal{C}_{\geq p}^{\supseteq ev}(T^E))$

be reduced to the decision problem *EventCover*. The following theorem gives the formal result.

THEOREM 4.6. *Let \mathcal{S} be a class of schemes. If for all (localized) p -rdb's over \mathcal{S} , *EventCover* can be decided in polynomial time, then `ProbableJCC` is in incremental polynomial time.*

4.2.2 Tractable Cases

By Theorem 4.6, the goal is to find an efficient decision algorithm for *EventCover*. Unfortunately, as we prove later, this problem is intractable in the general case. Now, we show that *EventCover* can be solved efficiently in the case of the three classes of database schemes that are defined in Section 2. Recall that when deciding whether the scheme of a p -rdb belongs to one of these classes, the special attribute E is ignored.

When the database scheme is a clique, solving *EventCover* is straightforward, because every join-consistent tuple set is also connected. Given \mathcal{P}, p, T^E and ev , we only need to verify that T^E contains all the events of ev and $\prod_{e \in ev} \Pr(e) \geq p$. If so, then $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E)$ is nonempty; otherwise, it is empty. Therefore, we get the following corollary of Theorem 4.6.

COROLLARY 4.7. *ProbableJCC is in incremental polynomial time for p -rdb's over $\mathcal{S}_{\text{clique}}$.*

We now consider a localized p -rdb over a γ -acyclic scheme. Note that in localized p -rdb's, distinct tuples of a JCC tuple set have distinct events. The algorithm `MSTPRBCVRS` of Figure 5 tests, in Line 1, whether T^E contains all the events of ev . If so, it constructs a set T_{\max}^E of $\mathcal{C}_{\geq 0}^{\supseteq ev}(T^E)$ that has the highest probability among all the elements of $\mathcal{C}_{\geq 0}^{\supseteq ev}(T^E)$. Clearly, $\mathcal{C}_{\geq p}^{\supseteq ev}(T^E)$ is nonempty if and only if $\Pr(T_{\max}^E) \geq p$.

Line 3 of Figure 5 constructs the set S consisting of all the ordinary relation schemes of the tuples of T^E . Line 4 computes the subset S_{ev} of S that comprises the relation schemes of tuples with events of ev . $\text{BD}(S)$ is the *Bachman diagram* of S . Recall that the nodes of $\text{BD}(S)$ are actually sets of attributes and, moreover, since S is γ -acyclic, $\text{BD}(S)$ is a tree. Line 5 constructs the subtree B_{ev} of $\text{BD}(S)$ that is reduced w.r.t. S_{ev} ; that is, all the relation schemes of S_{ev} appear among the nodes of B_{ev} , but no proper subtree of

Algorithm MSTPRBCVR(T^E, ev)

```

1: if  $ev \not\subseteq \pi_E(T^E)$  then
2:   fail
3:  $S \leftarrow \{\text{Attr}(t) \setminus \{E\} \mid t \in T^E\}$ 
4:  $S_{ev} \leftarrow \{\text{Attr}(t) \setminus \{E\} \mid t \in T^E \wedge t[E] \in ev\}$ 
5:  $B_{ev} \leftarrow$  the subtree of  $\text{BD}(S)$  that is reduced w.r.t.  $S_{ev}$ 
6:  $T_{\max}^E \leftarrow \{t \in T^E \mid t[E] \in ev\}$ 
7: for all maximal nodes  $M$  of  $B_{ev}$  do
8:   if  $\exists R \in S_{ev}$ , such that  $M \subseteq R$  then
9:      $T_M^E \leftarrow \{t' \in T^E \mid M \subseteq \text{Attr}(t')\}$ 
10:     $t_M \leftarrow$  a tuple  $t \in T_M^E$  with a maximal  $\text{Pr}(t[E])$ 
11:     $T_{\max}^E \leftarrow T_{\max}^E \cup \{t_M\}$ 
12: return  $T_{\max}^E$ 

```

Figure 5: Constructing a most probable tuple set T_{\max}^E of $\mathcal{C}_{\geq 0}^{\supset ev}(T^E)$ in a localized γ -acyclic p-rdb

B_{ev} has this property. A node M of B_{ev} is *maximal* if it is not properly contained in another node of B_{ev} .

In Line 6, T_{\max}^E is initialized to the set comprising the tuples of T^E that have events of ev . More tuples may have to be added to T_{\max}^E , in order to make it connected. This is done in the loop of Line 7 that iterates over all the maximal nodes M of B_{ev} . For each M , Line 8 tests whether S_{ev} has a relation scheme that contains M . If not, Lines 10 finds the tuple t_M with the highest probability among the tuples of T^E that include all the attributes of M . In Line 11, t_M is added to T_{\max}^E . The final value of T_{\max}^E is returned in Line 12. The following lemma shows the correctness of this algorithm. The proof of this lemma is rather intricate. The heart of this proof is showing that each cover of ev must contain, for each M , a distinct tuple that includes all the attributes of M .

LEMMA 4.8. *Consider a localized γ -acyclic p-rdb \mathcal{P} . Let ev be a set of events and $T^E \in \mathcal{C}(\mathcal{P})$. MSTPRBCVR(T^E, ev) returns, in polynomial time, a set $T_{\max}^E \in \mathcal{C}_{\geq 0}^{\supset ev}(T^E)$ that has the maximal probability, or no set at all if $ev \not\subseteq \pi_E(T^E)$.*

Thus, to solve *EventCover*, we run MSTPRBCVR and test whether the probability of the result is at least the threshold p .

THEOREM 4.9. *ProbableJCC can be solved in incremental polynomial time for localized γ -acyclic p-rdb.*

4.2.3 Intractability

As said earlier, the algorithm of Figure 3 is a reduction of the enumeration problem *ProbableJCC* to the decision problem *EventCover*. In the previous section, we have shown that for several classes of schemes, *EventCover* is tractable. However, the next theorem shows that, in general, there is no efficient algorithm for this decision problem. We prove this theorem by using two different reductions from *SetCover*.

THEOREM 4.10. *EventCover is NP-complete, even under either one of the following two restrictions. The p-rdb is localized or the scheme is a tree.*

In light of the above, it seems that our approach to solving *ProbableJCC* is rather limited, since it is efficient only

when *EventCover* is tractable. Rather surprisingly, the next theorem shows that our algorithm is optimal in the following sense. If any algorithm can efficiently solve *ProbableJCC* for all p-rdb over a given class of schemes, then *EventCover* is also tractable in that class of schemes. Thus, no algorithm can capture more tractable classes of schemes than the reduction of Figure 3. Essentially, the proof shows how *EventCover* can be decided in polynomial time by using any efficient algorithm for *ProbableJCC* (even one that runs in polynomial total time rather than incremental polynomial time).

THEOREM 4.11. *Let \mathcal{S} be a class of schemes. If there is a polynomial-total-time algorithm that solves ProbableJCC for all (localized) p-rdb over \mathcal{S} , then there is a polynomial-time algorithm that decides EventCover over those p-rdb.*

4.3 Other Tractable Cases

We now describe more common cases where *ProbableJCC* is tractable. These cases are characterized in terms of properties of distributions, rather than database schemes.

We say that a p-rdb \mathcal{P} has *equally probable events* if there exists a number $0 < p < 1$, such that the probability of each event e of \mathcal{P} is either 1 or p . For a number $0 < g < 0.5$, we say that \mathcal{P} has *g-bounded probabilities* if each event e of \mathcal{P} satisfies either $\text{Pr}(e) = 1$ or $g \leq \text{Pr}(e) \leq 1 - g$.

The following theorem says that if \mathcal{P} has either equally probable events or g -bounded probabilities (for some fixed g), then *ProbableJCC* can be solved with polynomial delay (regardless of the database scheme).

THEOREM 4.12. *Let $0 < g < 0.5$ be fixed. ProbableJCC can be solved with polynomial delay if*

- *The p-rdb has equally probable events; or*
- *The p-rdb has g-bounded probabilities.*

5. PROBABILISTIC PARTIAL JOINS

Consider an instance I of a p-rdb \mathcal{P} and a JCC tuple set T of I . We define $\bowtie(T)$ to be the tuple over all the ordinary attributes of \mathcal{P} that is obtained by joining the tuples of T and adding columns with the *null* value for the missing attributes. (The null value is denoted by \perp and we assume that it does not appear in \mathcal{P} .) Given an instance $I \in \mathcal{I}(\mathcal{P})$, we use $\mathcal{J}(I)$ to denote the set $\{\bowtie(T) \mid T \in \mathcal{C}(I)\}$. The result of partially joining the relations of \mathcal{P} is defined as $\mathcal{J}(\mathcal{P}) = \cup_{I \in \mathcal{I}(\mathcal{P})} \mathcal{J}(I)$. Clearly, $\mathcal{J}(\mathcal{P}) = \{\bowtie(\pi_{\perp E}(T^E)) \mid T^E \in \mathcal{C}(\mathcal{P})\}$. Note that $\mathcal{J}(\mathcal{P})$ is a set and, hence, does not have multiple copies of the same tuple; however, there are likely to be subsumptions (as defined below) among the tuples of $\mathcal{J}(\mathcal{P})$.

Example 5. Figure 2 shows the p-rdb \mathcal{P}_1 and the sets $\mathcal{C}(\mathcal{P}_1)$ and $\mathcal{J}(\mathcal{P}_1)$. As noted above, for each tuple u_i of $\mathcal{J}(\mathcal{P}_1)$, there is at least one JCC tuple set $T_j \in \mathcal{C}(\mathcal{P}_1)$, such that $u_i = \bowtie(\pi_{\perp E}(T_j))$. Specifically, u_1, \dots, u_6 are obtained from T_1, \dots, T_6 , respectively. The tuple u_7 of $\mathcal{J}(\mathcal{P}_1)$ is generated by three JCC tuple set of $\mathcal{C}(\mathcal{P}_1)$, i.e., $u_7 = \bowtie(T_7) = \bowtie(T_8) = \bowtie(T_9)$. Finally, u_8, u_9 and u_{10} are obtained from T_{10}, T_{11} and T_{12} , respectively.

Consider two tuples t_1 and t_2 over the set of all the ordinary attributes. We say that t_2 *subsumes* t_1 , denoted by $t_1 \sqsubseteq t_2$, if t_2 is equal to t_1 in every column where the latter is

nonnull. If, in addition, $t_2 \neq t_1$, then we say that t_2 *properly subsumes* t_1 . A *maximal* tuple of a set U is any tuple $t \in U$, such that t is not properly subsumed by another tuple of U . In Figure 2, for example, u_7 properly subsumes u_1, u_2, u_3 and u_8 . The maximal tuples of $\mathcal{J}(\mathcal{P}_1)$ are u_6, u_7, u_9 and u_{10} .

Consider a JCC tuple set T^E of \mathcal{P} and let $t = \bowtie(\pi_{\downarrow E}(T^E))$. We define the probability of t , denoted by $\Pr(t)$, as the probability that a random instance $I \in \mathcal{I}(\mathcal{P})$ satisfies $t \in \mathcal{J}(I)$. Note that $\Pr(t) \geq \Pr(T^E)$, but equality does not necessarily hold because several JCC tuple sets may yield the same t . In Figure 2, for example, the tuple $u_7 \in \mathcal{J}(\mathcal{P}_1)$ is in every $\mathcal{J}(I)$, such that the instance I has the tuples $\pi_{\downarrow E}(t_1)$ and either $\pi_{\downarrow E}(t_2)$ or $\pi_{\downarrow E}(t_3)$. Therefore,

$$\Pr(u_7) = 0.7 \cdot (1 - (1 - 0.7) \cdot (1 - 0.7)) = 0.637,$$

that is, the probability that the event e_1 and at least one of the events e_2 and e_3 occur.

Given a threshold $p \in [0, 1]$, we use $\mathcal{J}_{\geq p}(\mathcal{P})$ to denote the set $\{t \mid t \in \mathcal{J}(\mathcal{P}) \text{ and } \Pr(t) \geq p\}$. The set that consists of all the maximal tuples of $\mathcal{J}_{\geq p}(\mathcal{P})$ is denoted by $\mathcal{J}_{\geq p}^{\max}(\mathcal{P})$. Note that when all the probabilities are 1, the set $\mathcal{J}_{\geq p}^{\max}(\mathcal{P})$ is actually the full disjunction [14, 22, 17, 7, 5] of the relations of \mathcal{P} (assuming that column E is ignored).

In Figure 2, for example, the tuple u_7 is in $\mathcal{J}_{\geq 0.5}^{\max}(\mathcal{P}_1)$, but none of the three JCC tuple sets of $\mathcal{C}(\mathcal{P}_1)$ that generates u_7 is in $\mathcal{C}_{\geq 0.5}^{\max}(\mathcal{P}_1)$.

DEFINITION 5.1. *ProbablePJoin is the problem of computing $\mathcal{J}_{\geq p}^{\max}(\mathcal{P})$, given a p-rdb \mathcal{P} and a threshold $p \geq 0$.*

Next, we discuss the complexity of *ProbablePJoin*.

5.1 Tractability under Data Complexity

In this section, we analyze the data complexity of *ProbablePJoin*, that is, we consider a fixed database scheme S and the input is any p-rdb \mathcal{P} over S (and a threshold $p \geq 0$). Under data complexity, we can generate all the tuples of $\mathcal{J}(\mathcal{P})$ in polynomial time. Thus, if the probabilities are known, we can find the subset $\mathcal{J}_{\geq p}(\mathcal{P})$ and choose its maximal tuples. So, we now show that the probability $\Pr(t)$ of each tuple $t \in \mathcal{J}(\mathcal{P})$ can be computed in polynomial time.

Consider a p-rdb \mathcal{P} and a tuple $t \in \mathcal{J}(\mathcal{P})$. We define $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ to be the set of all tuples t' of \mathcal{P} , such that $\pi_{\downarrow E}(t') \sqsubseteq t$. Note that $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ is connected and every two tuples of $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ are join consistent, but (the set of events of) $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ is not necessarily feasible. Also observe that $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ has at most one tuple from each relation of \mathcal{P} (since two distinct tuples from the same relation of \mathcal{P} cannot agree on all their ordinary attributes).

We say that an instance I of \mathcal{P} is a *producer* of t if $t \in \mathcal{J}(I)$. Note that I is a producer of t if and only if there is a JCC tuple set T^E of \mathcal{P} , such that $T^E \subseteq \mathcal{J}_{\mathcal{P}}^{-1}(t)$, $\bowtie(\pi_{\downarrow E}(T^E)) = t$ and all the tuples of $\pi_{\downarrow E}(T^E)$ are in I .

A JCC subset T^E of $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ is *closed* if for every tuple $u \in T^E$, all the tuples of $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ that are created by the event $u[E]$ are in T^E ; that is, there is no event that generates some of the tuples of T^E as well as some other tuples that appear in $\mathcal{J}_{\mathcal{P}}^{-1}(t) \setminus T^E$.

For a given tuple $t \in \mathcal{J}(\mathcal{P})$, we partition all the producers of t into pairwise disjoint subsets as follows. Each partition corresponds to exactly one closed JCC subset T^E of $\mathcal{J}_{\mathcal{P}}^{-1}(t)$, such that $\bowtie(\pi_{\downarrow E}(T^E)) = t$. The partition for a particular T^E consists of all instances I of \mathcal{P} , such that I includes all

Algorithm TUPLEPROB(\mathcal{P}, t)

```

1:  $p_t \leftarrow 0$ 
2: for all closed subsets  $T^E \subseteq \mathcal{J}_{\mathcal{P}}^{-1}(t)$  do
3:   if  $T^E$  is a JCC subset and  $\bowtie(\pi_{\downarrow E}(T^E)) = t$  then
4:      $O \leftarrow \pi_E(T^E)$ 
5:      $v \leftarrow \prod_{e \in O} \Pr(e)$ 
6:     for all distributions  $x$ , such that  $x$  has some
       events in  $\mathcal{J}_{\mathcal{P}}^{-1}(t)$  but none in  $T^E$  do
7:       let  $e_1, \dots, e_k$  be all the events of  $x$  in  $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ 
8:        $v \leftarrow v \cdot (1 - \sum_{i=1}^k \Pr(e_i))$ 
9:      $p_t \leftarrow p_t + v$ 
10: return  $p_t$ 

```

Figure 6: Computing $\Pr(t)$

the tuples of $\pi_{\downarrow E}(T^E)$ and none of the tuples of $\pi_{\downarrow E}(\mathcal{J}_{\mathcal{P}}^{-1}(t) \setminus T^E)$. Note that the partitions are indeed pairwise disjoint, because each tuple of an instance I corresponds to a unique tuple of \mathcal{P} .

For a closed JCC subset T^E of $\mathcal{J}_{\mathcal{P}}^{-1}(t)$, the probability of the partition corresponding to T^E is the product of the following probabilities.

- $\Pr(e)$ for each event $e \in \pi_E(T^E)$.
- $1 - \sum_{i=1}^k \Pr(e_i)$ for each distribution x of \mathcal{P} , such that x has no event in T^E and e_1, \dots, e_k are all the events of x that appear in $\mathcal{J}_{\mathcal{P}}^{-1}(t)$.

Note that $1 - \sum_{i=1}^k \Pr(e_i)$ is the probability that the events of x that appear in $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ do not occur.

The probability of a tuple $t \in \mathcal{J}(\mathcal{P})$ is the sum of the above products over all closed JCC subsets T^E of $\mathcal{J}_{\mathcal{P}}^{-1}(t)$, such that $\bowtie(\pi_{\downarrow E}(T^E)) = t$. The algorithm is summarized in Figure 6. Under data complexity, $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ has a fixed number of subsets (since $\mathcal{J}_{\mathcal{P}}^{-1}(t)$ has at most one tuple from each relation of \mathcal{P}). Thus, we get the following lemma and theorem.

LEMMA 5.2. *TUPLEPROB(\mathcal{P}, t) returns $\Pr(t)$ in polynomial time under data complexity.*

THEOREM 5.3. *ProbablePJoin is solvable in polynomial time under data complexity.*

Recall that the probability of a JCC tuple set T^E is just the product of the events of $\pi_E(T^E)$. Not only is it easier to compute $\Pr(T^E)$, compared to the probability of a tuple $t \in \mathcal{J}(\mathcal{P})$, but there is also the following important property. If $\bar{T}^E \subseteq T^E$ then $\Pr(\bar{T}^E) \geq \Pr(T^E)$. There is no similar property for tuples of $\mathcal{J}(\mathcal{P})$; that is, $t_1 \sqsubseteq t_2$ does not necessarily imply that $\Pr(t_1) \geq \Pr(t_2)$. For example, in the set $\mathcal{J}(\mathcal{P}_1)$ of Figure 2, the tuple u_8 is properly subsumed by u_7 and also has a lower probability.

5.2 Query-and-Data Complexity

We now discuss the query-and-data complexity of the problem *ProbablePJoin*. In order to show intractable cases, we define the decision version of *ProbablePJoin*.

DEFINITION 5.4. *D-ProbablePJoin is the problem of determining, given a number n in unary representation, a p-rdb \mathcal{P} and a threshold p , whether $|\mathcal{J}_{\geq p}^{\max}(\mathcal{P})| \geq n$.*

Let C be any class of problems that contains P . The following proposition shows that if D -ProbablePJoin is C -hard, then ProbablePJoin cannot be solved efficiently (i.e., in polynomial total time) unless $C = P$.

PROPOSITION 5.5. *If ProbablePJoin is solvable in polynomial total time, then D-ProbablePJoin is solvable in polynomial time.*

The following theorem shows that under query-and-data complexity, ProbablePJoin becomes (highly) intractable. In particular, if ProbablePJoin can be computed in polynomial total time, then $P = P^{\#P}$. Furthermore, this result holds even if we assume the following. First, distinct tuples are produced by independent events, namely, all distributions are singletons and each event produces only one tuple. Second, the database scheme is a clique. Recall that $P^{\#P}$ is the class of problems that can be solved by polynomial-time machines that have an oracle to some $\#P$ -complete problem (e.g., the number of satisfying assignments of a CNF formula). Note that this class contains the whole polynomial hierarchy [23]. The proof of this theorem uses a reduction from $\#SetCover$, i.e., given a collection \mathcal{S} of subsets of a set \mathcal{X} , determine the number of subsets of \mathcal{S} that cover \mathcal{X} . This problem is known to be $\#P$ -complete [21].

THEOREM 5.6. *The following results hold even for p-rdbs over $\mathcal{S}_{\text{clique}}$ in which distinct tuples are produced by independent events.*

1. Computing $\Pr(t)$, given \mathcal{P} and t , is $\#P$ -hard.
2. D-ProbablePJoin is $P^{\#P}$ -hard.

5.2.1 Gamma-Acyclic Database Schemes

As the next theorem shows, assuming that the database scheme is γ -acyclic does not make ProbablePJoin any easier, even over localized p-rdbs. Note that the proof of Theorem 5.6 does not show hardness over γ -acyclic schemes. For this case, we use a reduction from the problem of computing the *permanent* of a 0/1 square matrix (or, equivalently, determining the number of complete matches in a bipartite graph). In [24], this problem is shown to be $\#P$ -complete.

THEOREM 5.7. *The following results hold even for localized p-rdbs over \mathcal{S}_γ .*

1. Computing $\Pr(t)$, given \mathcal{P} and t , is $\#P$ -hard.
2. D-ProbablePJoin is $P^{\#P}$ -hard.

Interestingly, the *nonemptiness* version of ProbablePJoin, i.e., determining whether $|\mathcal{J}_{\geq p}^{\max}(\mathcal{P})| \geq 1$, is in polynomial time for γ -acyclic (and not necessarily localized) p-rdbs. Thus, the importance of using the problem D -ProbablePJoin (rather than nonemptiness).

If there are no different tuple sets T_1^E and T_2^E , such that $\bowtie((\pi_{\perp E}(T_1^E)) = \bowtie((\pi_{\perp E}(T_2^E)))$, then $\bowtie(\cdot)$ is a probability-preserving bijection between $\mathcal{C}(\mathcal{P})$ and $\mathcal{J}(\mathcal{P})$. In this case, we can compute $\mathcal{J}_{\geq p}^{\max}(\mathcal{P})$ by generating the tuple sets of $\mathcal{C}_{\geq p}^{\max}(\mathcal{P})$ and applying the join to each one. Clearly, $\bowtie(\cdot)$ is a bijection if each relation of \mathcal{P} has a unique attribute that does not appear in any other relation. In γ -acyclic p-rdbs, $\bowtie(\cdot)$ is not necessarily a bijection. However, as we show next, it becomes a bijection if we add a reasonable requirement.

A database scheme $S = (\mathcal{A}_1, \dots, \mathcal{A}_k)$ is *containment free* if no relation scheme contains all the attributes of another relation scheme, that is, if $\mathcal{A}_i \subseteq \mathcal{A}_j$ ($1 \leq i, j, \leq k$), then $i = j$. We denote by $\mathcal{S}_\gamma^{\mathcal{C}}$ the set of all database schemes that are γ -acyclic and containment free.

LEMMA 5.8. *If \mathcal{P} is over $\mathcal{S}_\gamma^{\mathcal{C}}$, then $\bowtie(\cdot)$ is a probability-preserving bijection between $\mathcal{C}(\mathcal{P})$ and $\mathcal{J}(\mathcal{P})$.*

As a direct corollary of the above lemma and Theorems 4.9 and 4.12, we obtain the following result.

THEOREM 5.9. *The following hold for p-rdbs over $\mathcal{S}_\gamma^{\mathcal{C}}$. ProbablePJoin is solvable in incremental polynomial time if the p-rdb is localized. It is solvable with polynomial delay in the case of either equally probable events or g -bounded probabilities (for some constant $0 < g < 0.5$).*

Another interesting case is a tree scheme. In this case, the scheme is γ -acyclic, but $\mathcal{A} \subseteq \mathcal{A}'$ can hold for two relation schemes \mathcal{A} and \mathcal{A}' . When there are more than two relations (and the scheme is connected), the containment must be proper. We can reduce this case to the case of schemes in $\mathcal{S}_\gamma^{\mathcal{C}}$ by *eliminating* each relation R , such that the relation scheme of R is contained in another relation scheme R' . This is done as follows. Let t be a tuple of R . We can show that every tuple of $\mathcal{J}(\mathcal{P})$ that properly subsumes t must also subsume a tuple of R' . Thus, t is relevant to a tuple $\hat{t} \in \mathcal{J}(\mathcal{P})$ only if $t = \hat{t}$. Therefore, we check whether either $\Pr(t) < p$ or t is subsumed by a tuple t' of R' , such that $\Pr(t'[E]) \geq p$. If not, then we print t . After considering all tuples t of R , we remove R from \mathcal{P} . Consequently, we get the following theorem. Note that the lower bound is obtained by adapting the proof of Theorem 4.10 to tree schemes.

THEOREM 5.10. *The following hold for p-rdbs over $\mathcal{S}_{\text{tree}}$. D-ProbablePJoin is NP-complete and ProbablePJoin is solvable in incremental polynomial time for localized p-rdbs.*

6. CONCLUSION

We have studied the problem of maximally joining probabilistic relational data. In contrast to full disjunctions [14], there are two different interpretations of maximal answers: (1) maximal tuples that are produced by partial joins, and (2) maximal JCC tuple sets. In both cases, the query is meaningless unless the user specifies a threshold that determines the amount of certainty (i.e., probability) that the answers need to have. We have studied the query-and-data complexity of evaluating joins under the two semantics, namely, the problems ProbablePJoin and ProbableJCC.

We have shown that both problems have a tractable data complexity, yet are generally intractable under query-and-data complexity. In addition, we have identified several important cases where these problems are tractable. Table 1 summarizes (most of) our results regarding query-and-data complexity. We use the following notation. Poly_γ and Poly_- mean that the problem can be solved in incremental polynomial time and with polynomial delay, respectively. NPc (NP-complete) and $P^{\#P}$ h ($P^{\#P}$ -hard) refer to the decision versions of the two problems. Note that this table also includes results that are not discussed in the paper, e.g., for γ -acyclic schemes with either equal or bounded probabilities, ProbablePJoin is $P^{\#P}$ -hard.

p-rdb	<i>ProbableJCC</i>				<i>ProbablePJoin</i>				
	general	$\mathcal{S}_{\text{clique}}$	\mathcal{S}_{γ}	$\mathcal{S}_{\text{tree}}$	general	$\mathcal{S}_{\text{clique}}$	\mathcal{S}_{γ}	$\mathcal{S}_{\gamma}^{\leq}$	$\mathcal{S}_{\text{tree}}$
general	NPc	Poly_{γ}	NPc	NPc	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	NPc	NPc
localized	NPc	Poly_{γ}	Poly_{γ}	Poly_{γ}	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	Poly_{γ}	Poly_{γ}
equal-prob	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$
bounded-prob	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{P}^{\#\text{P}}_{\text{h}}$	$\text{Poly}_{\rightarrow}$	$\text{Poly}_{\rightarrow}$

Table 1: The query-and-data complexity of *ProbableJCC* and *ProbablePJoin*

Table 1 shows that *ProbablePJoin* is generally harder than *ProbableJCC* in the cases that we studied. It is unlikely that there are interesting cases where *ProbablePJoin* is easier than *ProbableJCC*, since we can solve the latter by using an algorithm for the former if we add a unique attribute to each relation. We have shown that there are efficient reductions between the two problems in several important cases, e.g., tree and containment-free γ -acyclic schemes. In future work, we intend to adapt the algorithm PROBABLEJCC so that it can directly solve *ProbablePJoin*. We already have an adaptation that works in special cases not shown in Table 1.

As for related work, various models of probabilistic relational databases have been proposed. Some of these models [11, 13, 10] are similar to ours in the sense that they associate an event (or just a probability) with each tuple. Other models [4, 18, 1] are different from ours, e.g., probabilities are associated with attributes [1, 18]. Except for [13], all of these models assume independence among the different relations. None of these models support mutually-exclusive events in different relations. In the probabilistic XML model presented in [20], children of the same XML element are associated with either independent or mutually-exclusive events. More importantly, as far as we know, no one has considered the notion of partial answers to queries over probabilistic data—a notion that is often crucial in practical scenarios that involve uncertainty. Furthermore, only data complexity of query evaluation has been studied. Under this measure, devising an acceptable algorithm for query evaluation (even when the user specifies a threshold) is just a matter of efficiently computing the probability of each answer. This is certainly not enough under query-and-data complexity. Ours is the first paper that analyzes query-and-data complexity in probabilistic databases.

Full-disjunctions [14, 22, 17, 7, 5] are closely related to our work. In fact, the basic technique of [5] is the starting point for developing our algorithm. In the case of full disjunctions, the two types of maximal answers coincide and can be computed efficiently [17] whereas, over probabilistic data, the evaluation of either type is generally intractable.

The notion of *approximate joins* is introduced in [8] as a generalization of full disjunctions. It is similar to our approach in that each tuple set has a score that is used, in conjunction with a threshold, in order to compute maximal tuple sets. But the algorithm of [8] works efficiently only when the score is the min function, whereas product is needed in our case. In fact, our tractable results generalize those of [8] (no intractability results were given there).

Interestingly, [22] shows that full disjunctions can be evaluated by outerjoins exactly over γ -acyclic schemes. But their results do not seem to be of any help in solving, over γ -acyclic schemes, the problems that we address.

7. REFERENCES

- [1] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5), 1992.
- [2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3), 1983.
- [3] P. A. Bernstein and N. Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10(4), 1981.
- [4] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In P. M. Stocker, W. Kent, and P. Hammersley, editors, *VLDB*, 1987.
- [5] S. Cohen, I. Fadida, Y. Kanza, B. Kimelfeld, and Y. Sagiv. Full disjunctions: Polynomial-delay iterators in action. In *VLDB*, 2006.
- [6] S. Cohen and Y. Sagiv. An abstract framework for generating maximal answers to queries. In *ICDT*, 2005.
- [7] S. Cohen and Y. Sagiv. An incremental algorithm for computing ranked full disjunctions. In *PODS*, 2005.
- [8] S. Cohen and Y. Sagiv. An incremental algorithm for computing ranked full disjunctions. To appear in *J. Comput. Syst. Sci.* (an extended version of [7]), 2006.
- [9] N. N. Dalvi, G. Miklau, and D. Suciu. Asymptotic conditional probabilities for conjunctive queries. In *ICDT*, 2005.
- [10] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.
- [11] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3), 1996.
- [12] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3), 1983.
- [13] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1), 1997.
- [14] C. A. Galindo-Legaria. Outerjoins as disjunctions. In *SIGMOD*, 1994.
- [15] D. Johnson, M. Yannakakis, and C. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27, 1988.
- [16] Y. Kanza, W. Nutt, and Y. Sagiv. Querying incomplete information in semistructured data. *J. Comput. Syst. Sci.*, 64(3), 2002.
- [17] Y. Kanza and Y. Sagiv. Computing full disjunctions. In *PODS*, 2003.
- [18] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3), 1997.
- [19] Y. E. Lien. On the equivalence of database models. *J. ACM*, 29(2), 1982.
- [20] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *VLDB*, 2002.
- [21] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4), 1983.
- [22] A. Rajaraman and J. D. Ullman. Integrating information by outerjoins and full disjunctions. In *PODS*, 1996.
- [23] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 21(2), 1992.
- [24] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8, 1979.
- [25] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.